PCS 3422

Organização e Arquitetura de Computadores II

Grupo F - Previsão de Desvio

Relatório Final

Bruno Brandão Inácio - 9838122

Pedro de Moraes Ligabue - 9837434

Pedro Henrique L. F. de Mendonça - 8039011

Vitor Tiveron de Almeida Santos - 9868085

Rodrigo Perrucci Macharelli - 9348877

07/11/2019

Sumário

1. Implementação

A implementação do módulo de previsão de desvio se baseia em uma tabela com um histórico de instruções de desvio já executadas, os endereços dos respectivos desvios, e uma previsão adquirida a partir dos resultados das instruções de desvio e uma máquina de estados.

O fluxo de funcionamento é o seguinte: quando uma instrução de desvio é carregada pela primeira vez, não estando na tabela, ela segue pelo *pipeline* como qualquer outra instrução, ou seja, o desvio não é previsto. Quando a instrução chega no estágio *Execute*, identificando-se que se trata de uma instrução de desvio, sinais de controle são ativados para que as informações do desvio sejam escritas na tabela na próxima subida do *clock*. O endereço da instrução, o endereço de desvio calculado pela ULA e o estado da instrução da tabela são incluídos. Mais explicações sobre a tabela estão incluídas na subseção 1.2.

Quando uma instrução é encontrada na tabela, ou seja, o valor do PC está registrado em sua memória, as suas saídas irão influenciar os próximos passos do *pipeline*. Se a tabela previr que haverá desvio, o próximo valor carregado no PC será o endereço destino armazenado em sua memória e uma *flag* de desvio previsto será passada adiante pelos estágios do *pipeline*. Caso seja previsto que não haverá desvio, o processamento prossegue normalmente.

No estágio de *Execute*, quando há a verificação da condição de desvio, caso a previsão tenha sido errada, os dois estágios anteriores, *Instruction Fetch* e *Instruction Decode*, são descartados.

1.1. Identificação de instruções de desvio

A identificação de uma instrução de desvio, necessária para que seja possível determinar quando deve ser feita a escrita na tabela de desvios, é feita através da Unidade de Controle. A UC possui uma *flag* de desvio, que sendo avaliada no estágio *Execute*, funciona como um *Write Enable* para a tabela.

1.2. Tabela com histórico de desvios

A tabela de desvios consegue armazenar 16 instruções de desvio, com seus respectivos endereços, os endereços de desvio e estados. O modelo adotado para a previsão de desvio é de contador saturado com 4 estados. Dessa maneira, tem-se as seguintes especificações para os estados:

Estado	**Descrição**	**Previsão**
00	Fortemente não tomado	Sem desvio
01	Fracamente não tomado	Sem desvio
10	Fracamente tomado	Desvio
11	Fortemente tomado	Desvio

Essa especificação determina que não haverá desvio para os estados “00” e “01”, e haverá desvio para “10” e “11”. A transição entre estados é feita a partir do resultado da instrução de desvio em questão, de forma que, havendo desvio, o estado muda em direção ao estado Fortemente Tomado e, não havendo desvio, o estado muda em direção ao estado Fortemente Não Tomado.

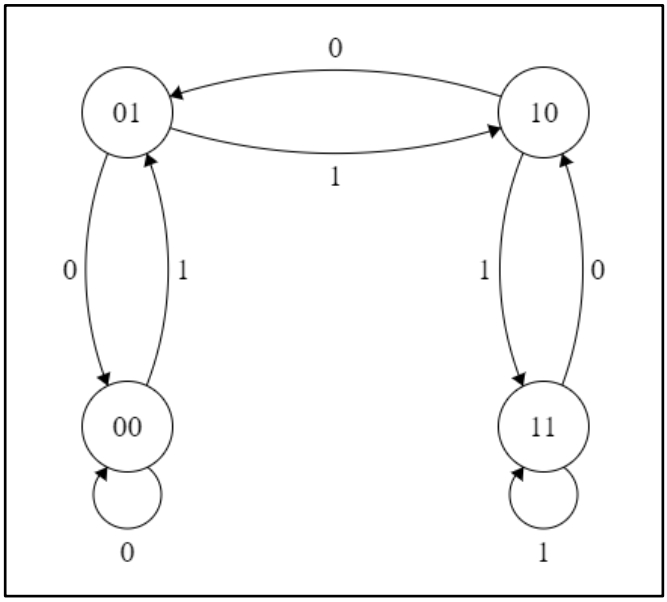


Figura 1 - FSM dos estados da tabela (0 - Sem desvio, 1 - Desvio)

A escrita é feita a partir dos dados presentes no estágio *Execute*. A tabela recebe como entrada um sinal de *enable* (equivalente ao sinal de *branch* da UC), um sinal com o endereço da instrução de desvio (equivalente ao sinal PC), o endereço de desvio (calculado no *Execute*) e o resultado do desvio (tomado ou não). Caso o endereço do PC seja encontrado na tabela, o estado é atualizado utilizando o resultado. Caso contrário, é feita uma nova escrita.

Nova escritas na tabela são feitas a partir de um ponteiro que sempre aponta para a parte mais antiga (ou a parte vazia) da tabela, onde deve ser feita a escrita. Quando há uma nova entrada na tabela, esse ponteiro é incrementado, de forma a circular pela tabela. Isso potencialmente gera uma perda, uma vez que uma linha da tabela ser antiga não equivale a ela não estar sendo utilizada. No entanto, como decisão de projeto e se baseando no livro *Computer Organization and Design* [1], optou-se por fazer esse tipo de acesso pseudo aleatório, uma vez que o ganho com o acesso LRU (*Least Recently Used*) apresentaria um ganho muito baixo (em torno de 10% [1]) para o nível de complexidade que acrescentaria.

Para a previsão de fato, verifica-se se o valor do PC no *Instruction Fetch* está na tabela. Caso o valor esteja e a previsão seja que haverá desvio, a saída da

tabela será o endereço de desvio junto com um sinal de controle responsável por controlar o multiplexador de entrada no PC. Dessa maneira, na próxima subida do *clock*, o endereço de desvio é carregado no PC. Além disso, uma *flag* de previsão de desvio é adicionada ao fluxo do *pipeline*. Isso conclui a previsão de desvio.

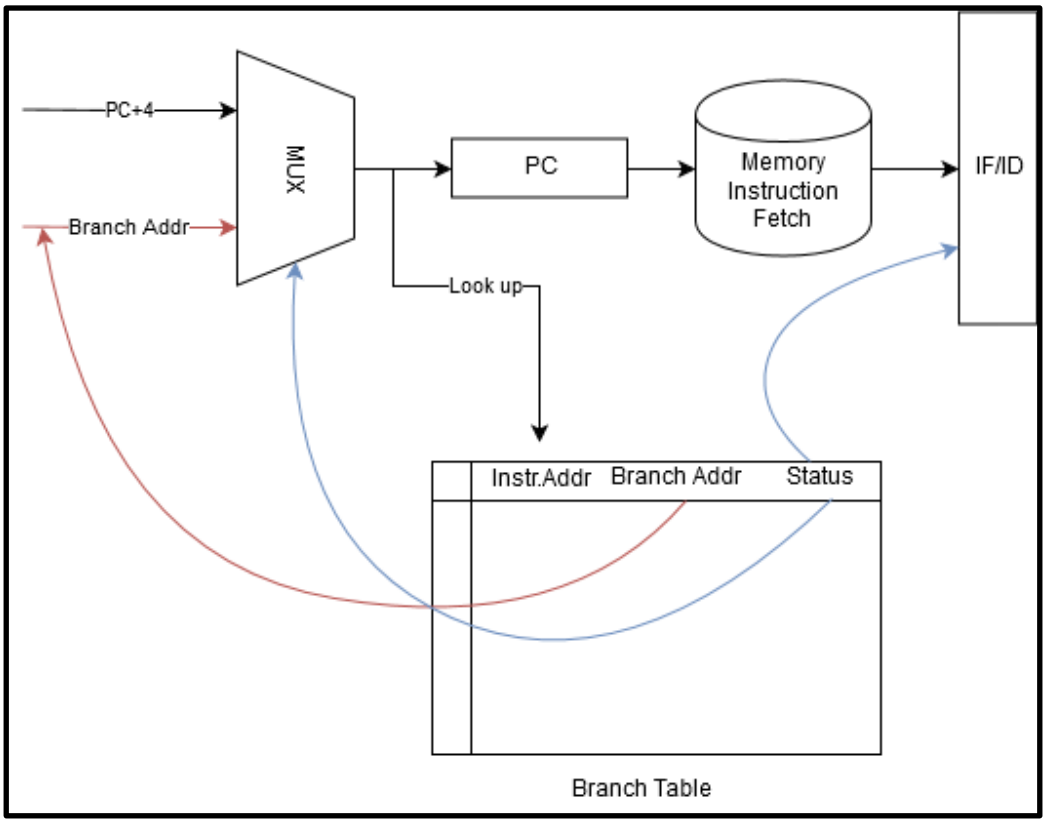
1.3. Descarte do Pipeline

No caso de uma previsão errada, é necessário descartar as instruções que já estão no *pipeline* em estágios anteriores ao de verificação (*Execute*). Os estágios que devem ser descartados são o *Instruction Fetch* e o *Instruction Decode*. Para fazer isso é adicionado um multiplexador antes dos registradores de estágio do IF e do ID, que usando o sinal de descarte como controle, seleciona entre o resultado regular da instrução e um sinal de zeros, representando um NOP. Dessa maneira, na subida do *clock*, o valor correto do PC é carregado (vindo do *Execute* no ciclo anterior) e os dois estágios seguintes, previstos erroneamente, estão zerados, impedindo que seja feita alguma alteração na memória ou os registradores.

2. Integração com o pipeline

2.1. Instruction Fetch

Internamente no estágio *Instruction Fetch*, são feitas três alterações principais. Em primeiro lugar, o sinal do PC é utilizado como uma entrada na tabela de desvios. Em segundo lugar, na entrada 0 do multiplexador de desvio já existente é adicionado outro multiplexador, que, usando a previsão de desvio como seletor, seleciona entre PC+4, valor caso não haja desvio, e o endereço de desvio vindo da tabela. Por último, a *flag* de previsão é passada para o registrador de estágio do *pipeline*, pois será utilizada no estágio *Execute*.



2.2. Instruction Decode

No estágio *Instruction Decode* a única alteração é que a *flag* com a previsão é passada para para o registrador de estágio do *pipeline*, para ser utilizada no estágio seguinte.

2.3. Execute

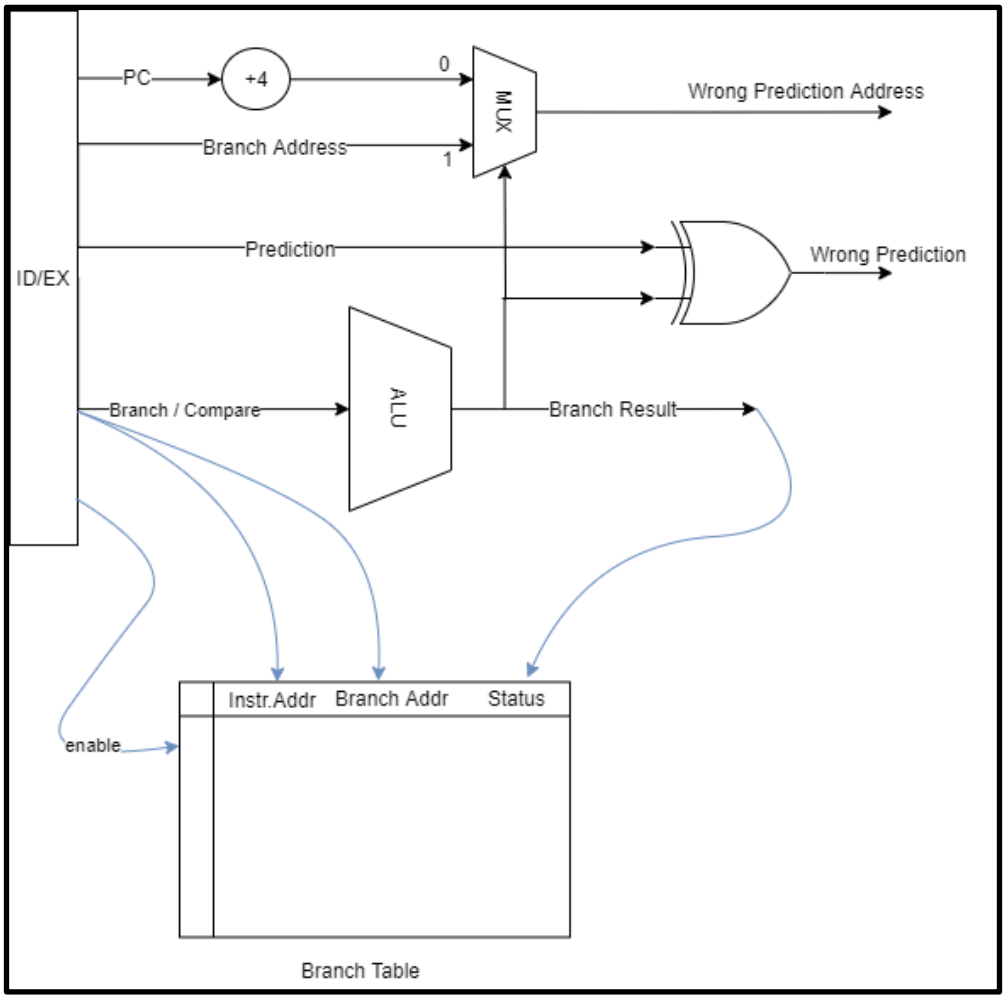
No estágio *Execute*, a alteração principal é a passagem da lógica de desvio que antes estava no estágio seguinte (*Memory Access*) para este estágio. Para isso, é necessário utilizar os sinais de controle *Branch on Zero* (BZ) e *Branch on Not Zero* (BNZ), o resultado da ULA (*Zero*) e a previsão de desvios para determinar o que escrever na tabela e se o *pipeline* deve ser descartado ou não. O sinal *Branch* funciona como o identificador de uma instrução de desvio e o *Write Enable* da tabela de desvios, o sinal *Zero* gera o resultado do desvio, que é usado com a previsão para determinar se o *pipeline* deve ser descartado e é usado para determinar o novo estado da instrução de desvio na tabela. Dessa maneira, a lógica (em pseudocódigo) fica como segue:

- Write Enable ← BZ **or** BNZ
- Branch Result ← **(**Zero **and** BZ**) or (**Zero**'** **and** BNZ**)**
- Discard Pipeline ← Branch Result **xor** Prediction

Além disso, os endereços certos do PC devem estar disponíveis, uma vez que a previsão errada demanda uma alteração no endereço das instruções. O endereço de desvio é a saída da *ULA* de endereços. Já o endereço sem desvio é a saída um somador separado, que soma o valor do PC que vem do registrador de estágio, equivalente ao endereço da instrução de desvio, com 4, de forma análoga ao que é feito no *Instruction Fetch*.

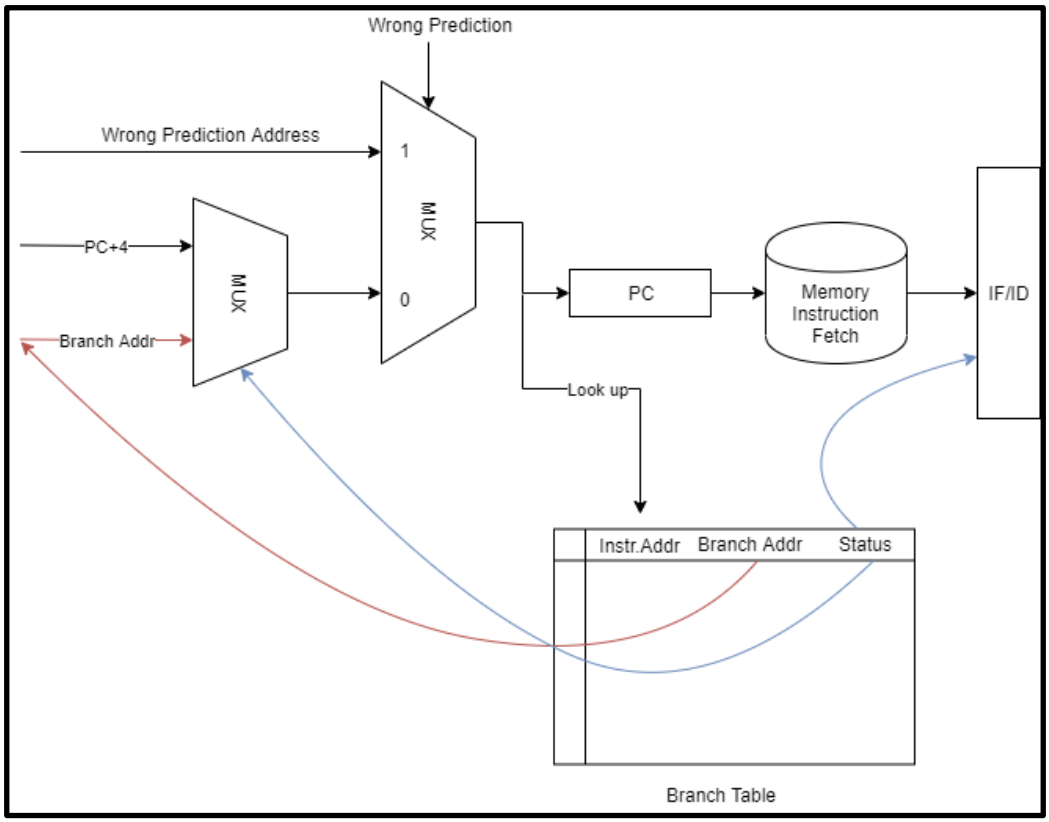
O carregamento desses endereços corretos é feito através de multiplexadores, que usam os mesmo sinais antes descritos da seguinte maneira:

- Internamente no *Execute*, o endereço a ser enviado para o IF depende do **Branch Result**. Se o desvio foi tomado, o endereço é a saída da ULA de endereços. Se não foi tomado, o endereço é o PC que está no registrador de estágio somado de 4 (**PC_{exe}+4**). Isso é feito utilizando um multiplexador com esses dois endereços como entrada e o **Branch Result** como seletor.



- No IF, se a previsão estiver errada, o valor carregado no PC será o vindo do Execute. Se estiver certa, o valor carregado no PC vem do próprio IF, sendo ou **PC+4** ou o endereço previsto pela tabela. O sinal de controle no

multiplexador nesse caso é o **Discard Pipeline** [≙ `Branch Result XOR Prediction`].



2.4. Memory Access

A mudança no estágio de Memory Access é relativa à passagem da lógica de desvio deste estágio para o anterior.

3. Testes

Para verificação do correto funcionamento dos componentes do projeto foram feitos testes tanto na tabela de previsão de desvios quanto na integração com o pipeline desenvolvido pelo grupo por ele responsável.

3.1. Tabela

Os testes na tabela de previsão de desvios foram realizados com auxílio de uma bancada de testes nos quais foram simulados estímulos correspondentes a execução de instruções de desvio pelo *LEGv8*. Para simplificação do processo de teste e interpretação dos resultados, utilizou-se uma versão reduzida da tabela, com apenas 4 linhas e endereços de 4 bits.

Inicialmente efetuamos uma leitura na tabela, neste momento vazia, no endereço **0x6** retornando zero tanto para o valor da predição quanto para o valor do endereço de desvio. Em seguida faz-se uma escrita no endereço **0xE** supondo que ocorreu desvio para o endereço 0x6. O valor **0xE** é inserido na tabela e associado ao estado **10** (fracamente tomado), uma vez que foi um desvio realizado.

Faz-se uma leitura na tabela referente ao endereço **0xE** inserido na etapa anterior, retornando os valores **0x6** para o endereço de branch e **1** para o predict, como esperado, já que este endereço está em um estado favorável à realização de branch. Realiza-se uma escrita na tabela referente ao endereço **0x7** com endereço de branch **0xD** supondo que o desvio não foi tomado. O endereço **0x7** é inserido na tabela e associado ao estado **01** (fracamente não tomado).

Neste ponto tenta-se efetuar uma escrita na tabela no endereço **0xE**, já nela inserido e referenciando o endereço de desvio **0x6**, supondo a ocorrência de um desvio para o endereço **0xC**. Neste caso o endereço destino da tabela se mantém inalterado, já que o endereço anteriormente associado a **0xE** na tabela é diferente

do endereço da nova ocorrência. Há uma mudança no estado desta linha para **11** (fortemente tomado), já que houve desvio.

Testa-se então o comportamento do algoritmo quando efetua-se uma escrita no endereço **0x7**, já contido na tabela, novamente não realizando o desvio para o mesmo endereço destino já associado. Nota-se a alteração do estado referente a este endereço de **01** (fracamente não tomado) para **00** (fortemente não tomado).

A situação oposta a anterior é simulada ao escrever no endereço **0xE**, que se encontra no estado **11**, a não ocorrência de desvio para o mesmo endereço destino, alterando seu estado para **10**.

Uma sequência de operações foi simulada sobre os dois endereços registrados na tabela de maneira ambos passassem por todos os possíveis estados, demonstrando o funcionamento esperado da máquina de estados.

Verifica-se o funcionamento da tabela no caso especial em que todas as suas linhas estão ocupadas e ocorre a uma nova instrução de desvio, ainda não listada. Neste caso observa-se que o elemento mais antigo da tabela é substituído, como esperado, uma vez que não se implementa a técnica *LRU*.

Por fim, verifica-se a funcionalidade do sinal de reconhecimento de instrução de desvio, que caso esteja em nível baixo deve desabilitar qualquer tipo de alteração na tabela, atuando como um sinal de enable de escrita. Mantendo-o em zero e simulando a ocorrência de um desvio, percebe-se que a tabela não se altera, de forma esperada.

3.2. Pipeline

Para os testes do pipeline, foi feito um código simples com um pequeno loop para observar o comportamento do pipeline na presença de instruções de desvio condicional (CBZ), desvio incondicional (branch) e de instruções que não são de desvio. O seguinte código foi utilizado:

```
    x"91000C02", -- 0: addi x2, x0, #3
    x"00000000", -- 4: NOP
    x"00000000", -- 8: NOP
    x"00000000", -- C: NOP
    x"b4001042", --10: CBZ to #228 when x2 = 0
    x"D1000442", --14: sub x2, x2, #1
    x"00000000", --18: NOP
    x"00000000", --1C: NOP
    x"00000000", --20: NOP
    x"00000000", --24: NOP
    x"00000000", --28: NOP
    x"00000000", --2C: NOP
    x"00000000", --30: NOP
    x"17fffff5", --34: branch para a posição #18
    x"91000C01", --38: addi x1, x0, #3
    x"91000C01", --3C: addi x1, x0, #3
    x"91000C01", --40: addi x1, x0, #3
    x"91000C01", --40: addi x1, x0, #3
    x"00000000", --48: NOP
    x"00000000", --4C: NOP
    x"00000000", --50: NOP
    x"00000000", --54: NOP
    x"00000000", --58: NOP
    x"00000000", --5C: NOP
```

Para observar o comportamento do pipeline ao longo da execução da simulação foi utilizado o programa ModelSim. No software é possível selecionar sinais desejados e observá-los ao longo dos ciclos de execução do pipeline.

Inicialmente, observamos um caso onde a previsão se mostra incorreta. A instrução branch na posição #34 da memória é considerada como não tomada e o pipeline carrega as instruções na sequência da memória.

Observa-se que o PC, representado por *pc_out* está na posição #3C na imagem e carrega a instrução (*instr_if*) 91000C01, que não deveria ser carregada, visto que está após a instrução de branch, da posição #34.

O sinal *wrong_prediction* é ativado, indicando que a previsão foi incorreta e, portanto, os dois primeiros estágios do pipeline precisam ser esvaziados e o PC precisa ser atualizado.

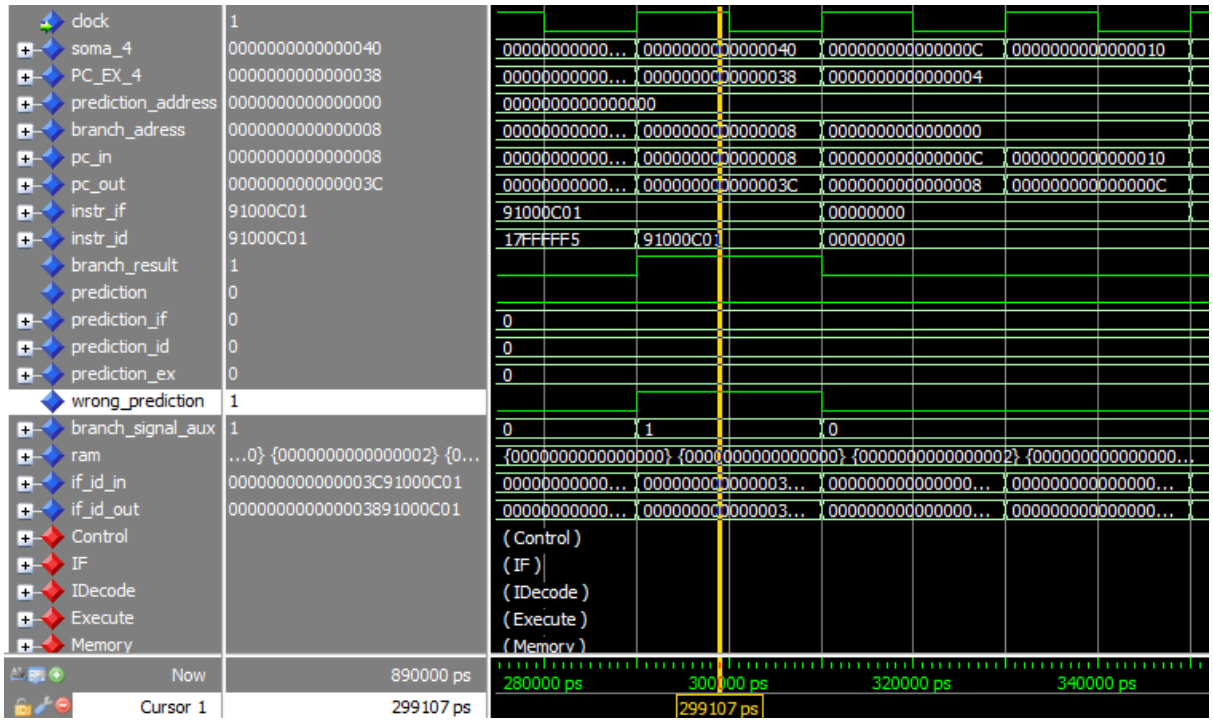


Figura 3.2.1 - Previsão incorreta é percebida

Após isso, o PC é corrigido e os dois primeiros estágios do pipeline são eliminados. Isso pode ser observado pelos sinais *if_id_in* e *if_id_out* que representam, respectivamente, a entrada do registrador do estágio de Decode (e, portanto, a saída do registrador do estágio de Fetch) e a saída do registrador do estágio de Decode.

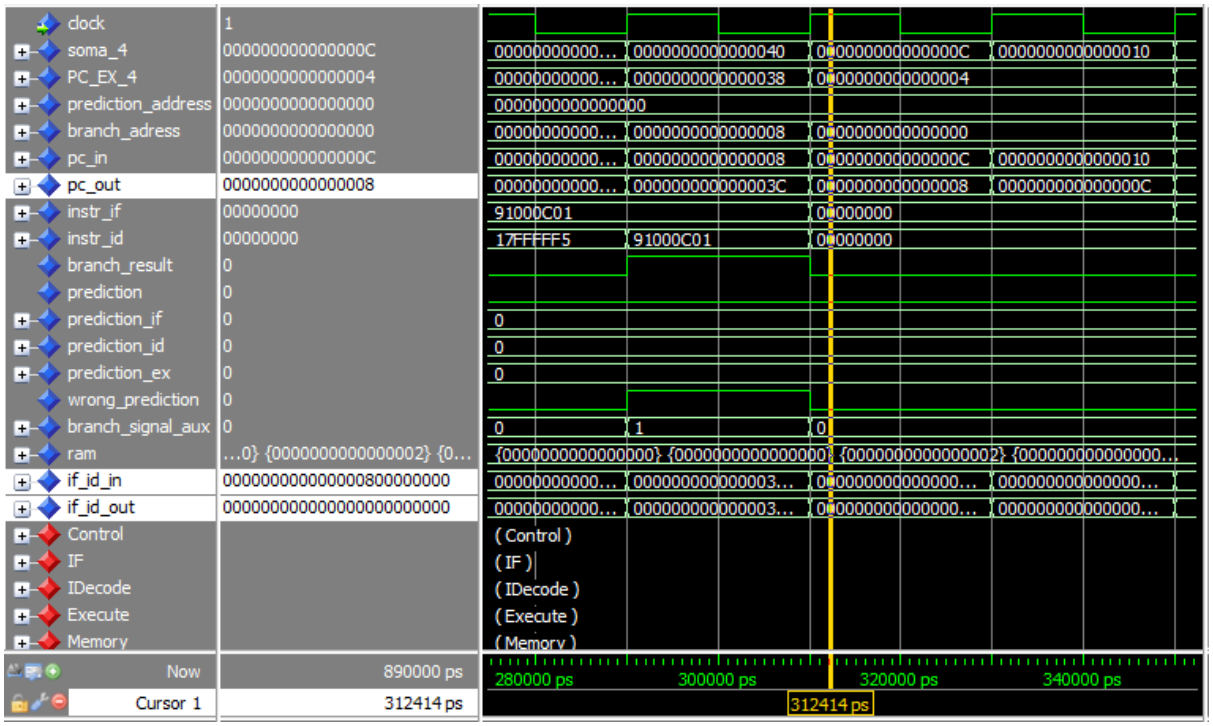


Figura 3.2.2 - PC corrigido e os dois primeiros estágios do pipeline esvaziados

Na próxima vez que o loop passa por este trecho do código, pode-se perceber que que é feita a predição e o endereço do PC é atualizado logo que a instrução de branch passa do estágio de Fetch.

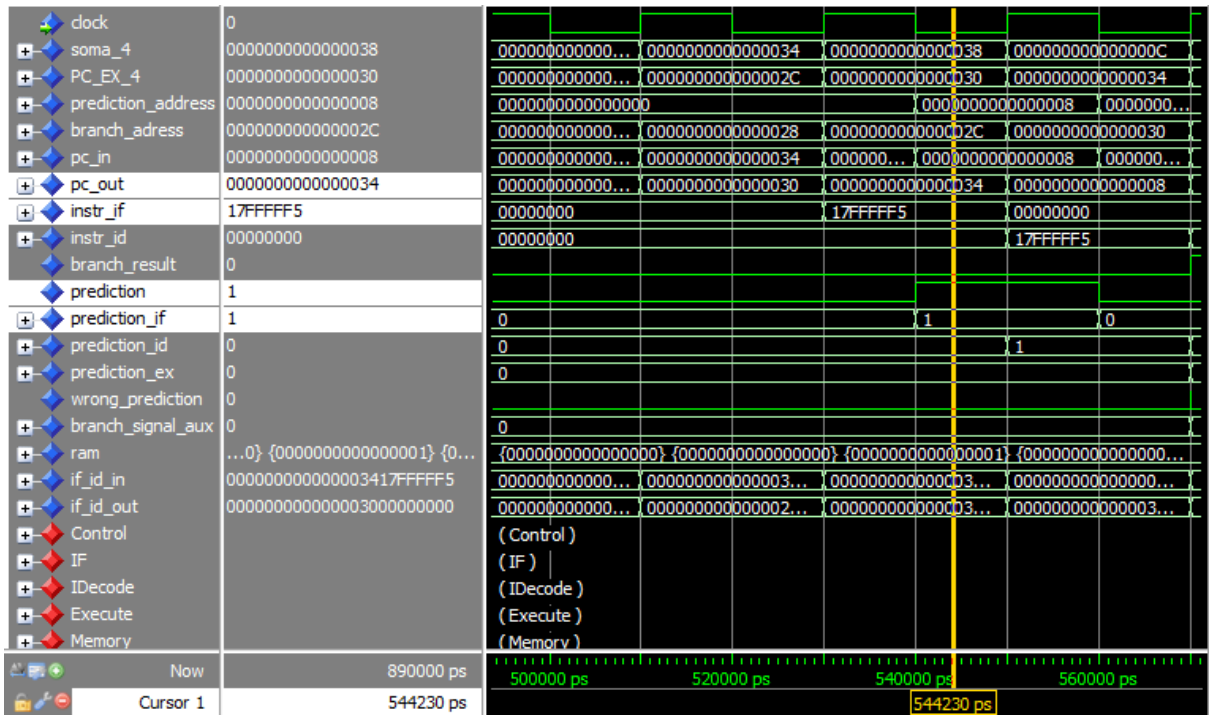


Figura 3.2.3 - Previsão é observada no estádio de Fetch.

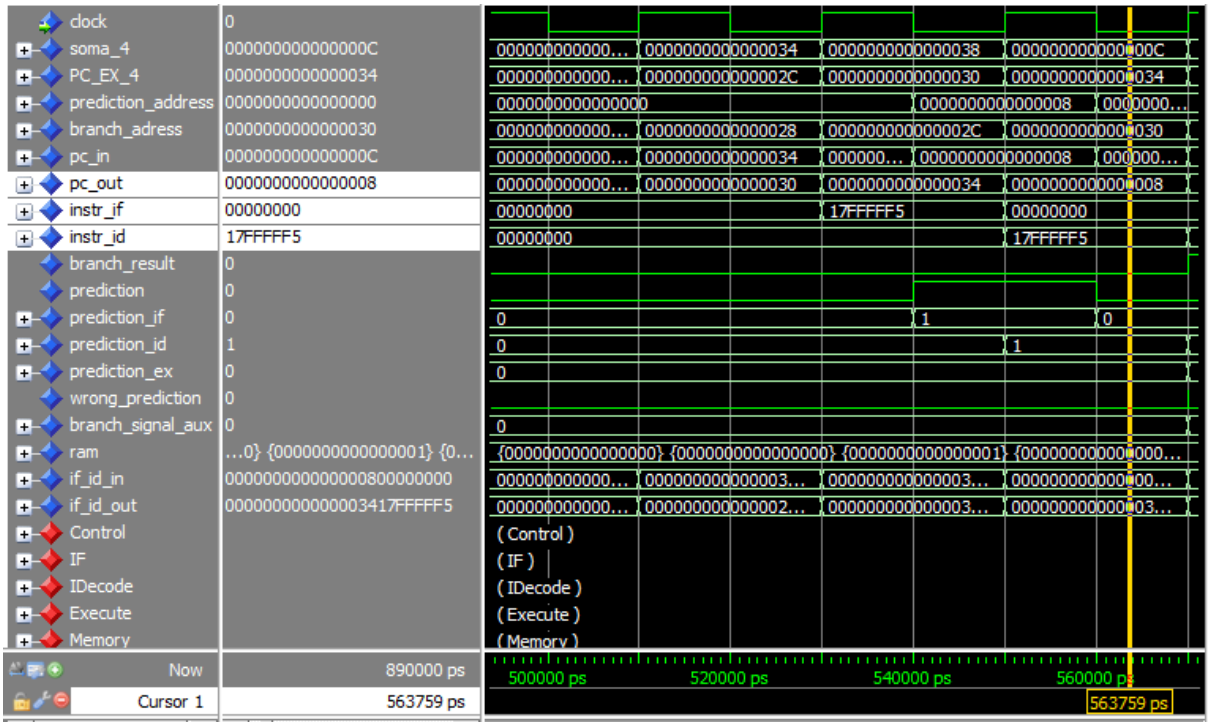


Figura 3.2.4 - PC é atualizado para a posição do branch e carrega a instrução correta.

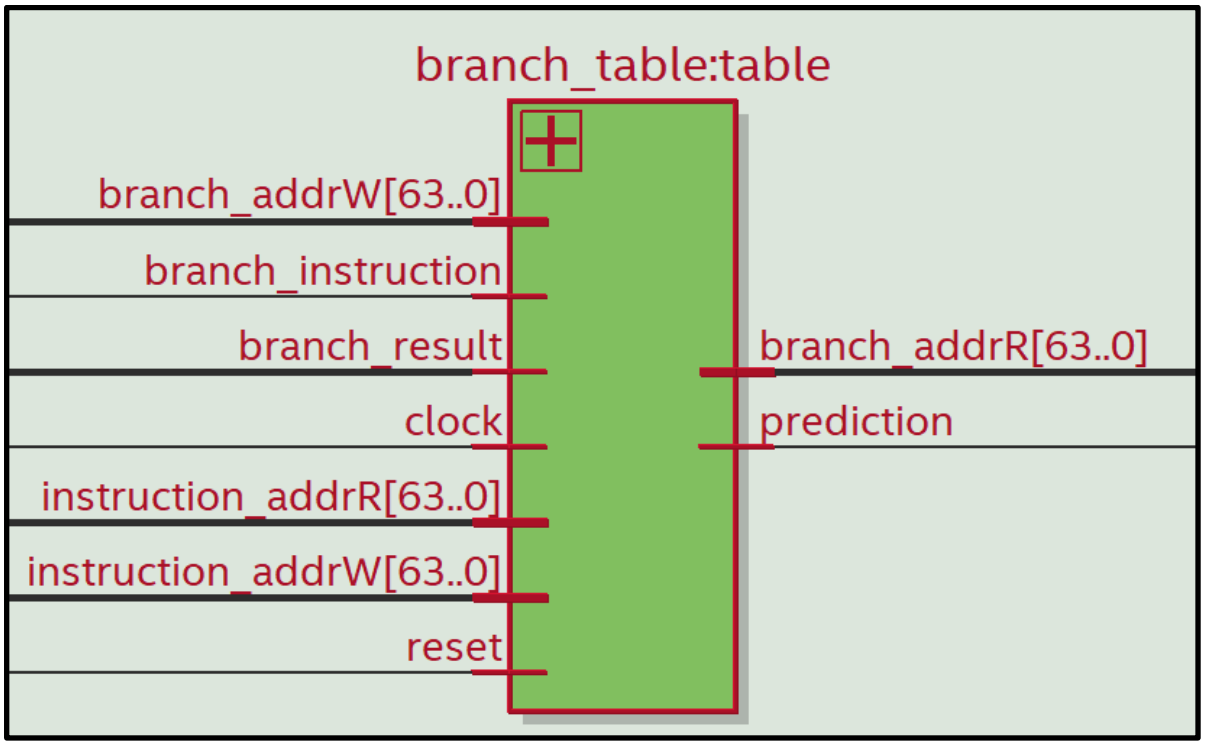
O código em questão, com 3 iterações, precisou de 890.000 ps para ser executado. No pipeline original, sem previsão de desvio, foram necessários 1.050.000 ps. O speedup foi, portanto: 1.179.

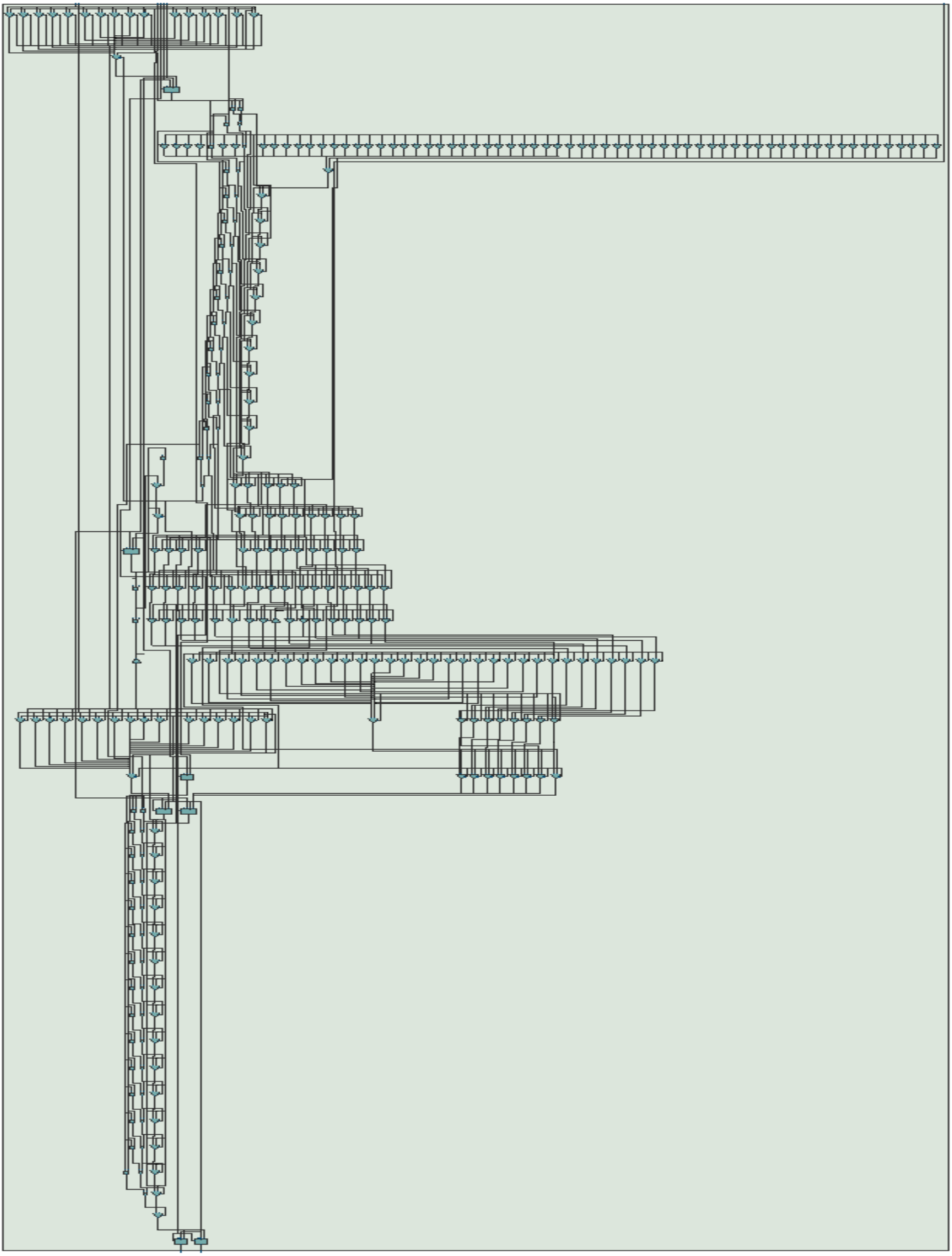
Uma versão do mesmo código, mas com 64 iterações precisou de 15.530.000 ps para ser executado, enquanto no pipeline sem previsão de desvio, foram necessários 20.250.000 ps. Gerando um speedup de 1.303.

O ganho de desempenho proporcionado pela previsão de desvio depende altamente do perfil do programa em execução, como por exemplo da quantidade e tamanho de loops e verificações. Por conta disso, é difícil estimar um valor exato para o ganho de desempenho.

4. RTL

Os seguintes RTLs gerados pelo software Quartus apresentam a estrutura da tabela desenvolvida.





5. Referências

[1] Patterson e Hennessy, Computer Organization and Design: The Hardware/Software Interface, Capítulo 5.8: A Common Framework For Memory Hierarchy, Página 457.